Tkinter

Comment créer une interface utilisateur graphique en Python

© Jacques Duma

9 février 2013

Ce document est un mode d'emploi simplifié du module `Tkinter` de Python qui explique comment construire une interface graphique rapidement et simplement.

Il ne s'agit pas d'un cours sur la programmation par objets, et seuls les concepts vraiment indispensables à la compréhension sont présentés à partir d'exemples simples.

Table des matières

1 Exemple élémentaire avec `Tkinter`: `"B-A-BA-Tkinter.py"` 3

2 Conceptualisation de l'interface graphique 4
- 2.1 Module `Tkinter` . . . 5
- 2.2 Objet, Classe, Constructeur, Instance, Propriété, Méthode . . . 5
- 2.3 Callback . . . 5
- 2.4 Layout Manager . . . 6
- 2.5 Boucle des événements . . . 6

3 Objets `Tkinter`, Widgets 6
- 3.1 Fenêtre principale : `Tk` . . . 6
- 3.2 Étiquette : `Label` . . . 7
- 3.3 Widget actif ou passif . . . 7
- 3.4 Bouton : `Button` . . . 8
- 3.5 Champ de saisie de texte : `Entry` . . . 8
- 3.6 Case à cocher : `Checkbutton` . . . 9
- 3.7 Cercle d'option : `Radiobutton` . . . 9
- 3.8 Zone de dessin : `Canvas` . . . 10
- 3.9 Cadre : `Frame`, `LabelFrame` . . . 10

4 Fenêtres de dialogues auxiliaires 10

5 Construction d'une interface complexe 11
- 5.1 Avec la méthode `pack`: `"Pack-Tkinter.py"` . . . 11
- 5.2 Avec la méthode `grid`: `"Grid-Tkinter.py"` . . . 14

6 Informations complémentaires 16

A Dimension **17**

B Encadrement **17**

C Couleur **17**

D Dessiner dans un `Canvas:` `"Canvas-Tkinter.py"` **18**
D.1 Ligne avec `create_line` 18
D.2 Polygone avec `create_polygon` 18
D.3 Rectangle avec `create_rectangle` 19
D.4 Ellipse avec `create_oval` 19
D.5 Arc avec `create_arc` 20

Remarque : Tous les exemples de code Python proposés dans cette documentation ont été testés sous le système Mac OS. Les images d'interfaces graphiques affichées ont un aspect qui pourra être légèrement différent sous les systèmes Linux ou Windows.

Les exemples proposés dans ce document sont disponibles à la rubrique « Interface » du site ISN, « Spécialité Informatique et Sciences du Numérique »

`http://mathmac.free.fr/-interface.html`

Ces exemples sont enregistrés dans les quatre fichiers suivants :

`"B-A-BA-Tkinter.py"` `"Canvas-Tkinter.py"` `"Pack-Tkinter.py"` `"Grid-Tkinter.py"`

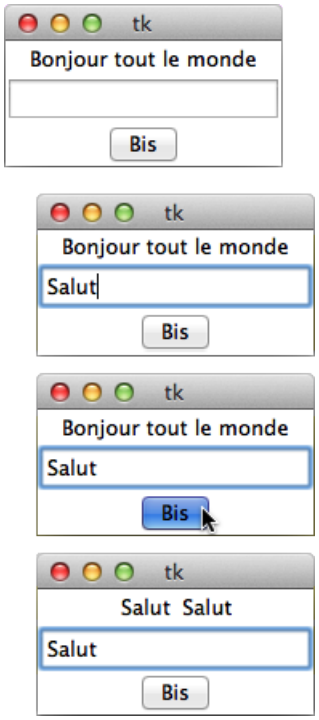
1 Exemple élémentaire avec `Tkinter` : `"B-A-BA-Tkinter.py"`

Une *Interface Utilisateur Graphique* (*Graphical User Interface* ou GUI en anglais) est un programme qui permet à l'utilisateur d'un ordinateur de fournir des informations, de les traiter ou de les consulter sur l'écran en utilisant le clavier ou la souris pour effectuer des interactions.

Nous allons d'abord proposer quelques lignes de code qui permettent d'afficher une fenêtre dans laquelle l'utilisateur pourra taper un texte court dans un champ de saisie, et ensuite, par un clic sur le bouton "Bis" provoquer l'affichage du texte saisi répété deux fois.

Cet exemple très simple présente les concepts de base mis en œuvre dans une Interface Utilisateur Graphique. Les image ci-dessous montre l'aspect de l'interface à l'ouverture, puis à la suite des interactions de l'utilisateur : (*ce code est dans le fichier* `B-A-BA-Tkinter.py`)

```
1  import Tkinter

2  fenetre = Tkinter.Tk()
3  etiquette = Tkinter.Label(fenetre)
4  etiquette["text"] = "Bonjour tout le monde"
5  champ = Tkinter.Entry(fenetre)
6  bouton = Tkinter.Button(fenetre, text="Bis")

7  def faireBis():
8      texte = champ.get()
9      texte = texte + " " + texte
10     etiquette["text"] = texte

11 bouton["command"] = faireBis

12 etiquette.pack()
13 champ.pack()
14 bouton.pack()

15 fenetre.mainloop()
```



La première ligne du code indique qu'il faut charger `Tkinter`, un module Python qui va nous permettre de définir notre interface.

À la ligne 2, on affecte la variable `fenetre` avec un objet de *classe* `Tk`, c'est à dire un objet interne Python qui représente la fenêtre principale. Pour cela on appelle la fonction `Tkinter.Tk` qui est un *constructeur* d'objet de classe `Tk`. La notation pointée précise que le constructeur `Tk` fait partie du module `Tkinter`.

On va maintenant définir les *widgets*, c'est à dire les éléments présents dans la fenêtre.

À la ligne 3, on affecte la variable `etiquette` avec un objet de classe `Label`, c'est à dire une étiquette. On remarque ici que l'appel du constructeur `Tkinter.Label(fenetre)` a pour argument `fenetre`, ce qui précise que l'étiquette est placée dans la fenêtre. On dit que l'objet `fenetre` est le *parent* de l'objet `etiquette`.

À la ligne 4, on fixe la valeur d'une *propriété* d'un objet de classe `Label`. Cette instruction précise que la propriété `text` de l'objet `etiquette` a pour valeur `"Bonjour tout le monde"`.

À la ligne 5, on affecte la variable `champ` avec un objet de classe `Entry`, c'est à dire champ de saisie de texte qui aura lui aussi `fenetre` pour parent.

À la ligne 6, on affecte la variable `bouton` avec un objet de classe `Button`, c'est à dire un bouton cliquable qui aura lui aussi `fenetre` pour parent. On remarque que cette instruction a un second argument `text="Bis"` qui fixe dès sa création la propriété `text` de l'objet.

Lors d'un clic de l'utilisateur dans le bouton, une certaine tâche doit être exécutée.
Aux lignes 7 à 10, on définit une fonction `faireBis` pour effectuer cette tâche :

- à la ligne 8, on applique la *méthode* `get` à `champ` qui est un objet de classe `Entry` ce qui retourne le texte contenu dans le champ de saisie. Ce texte est affecté à la variable `texte`.
- à la ligne 9, on duplique ce texte dans la variable `texte`.
- à la ligne 10, on modifie la propriété `text` de l'objet `etiquette`
  le texte saisi par l'utilisateur sera ainsi affiché deux fois dans l'étiquette

À la ligne 11, on fixe la valeur de la propriété `command` le l'objet de classe `Button` en lui donnant pour valeur la fonction `faireBis`. On précise ainsi qu'un clic de l'utilisateur sur le bouton provoquera l'appel de cette fonction aussi appelée *callback*.

Attention : Ici, l'interface graphique n'existe pas encore. On ne dispose que d'une représentation interne des objets nécessaires à sa création.
L'objet `fenetre` a trois fils, les widgets `etiquette`, `champ` et `bouton`.

Aux lignes 12 à 14, on envoie le message `pack()` aux différents widgets, ce qui a pour effet de les placer dans la fenêtre. Le gestionnaire de position ou *Layout Manager* fixe ainsi automatiquement, à l'aide de la méthode `pack`, les paramètres géométriques qui définissent l'aspect graphique de la fenêtre.

Attention : Ici, une représentation géométrique interne de l'interface graphique existe mais rien n'est encore visible sur l'écran.

À la ligne 15, on applique la méthode `mainloop` à l'objet `fenetre`.

Lors de l'exécution de ce programme Python, la fenêtre va être affichée (comme sur la figure ci-dessus) et le programme va tourner en boucle dans l'attente des événements provoqués par l'utilisateur avec la souris ou le clavier. Les différents événements pris en compte par ce programme sont :

- le clic dans le champ de saisie, ce qui sélectionne ce champ
- les frappes de caractères sur le clavier quand le champ de saisie est sélectionné
- le clic sur le bouton
- le clic sur la case de fermeture de la fenêtre

Ce dernier événement provoque la disparition de la fenêtre et la fin du programme.

2 Conceptualisation de l'interface graphique

Chaque ligne de code de ce programme Python introduit des notions spécifiques à la programmation événementielle par objets.

Nous allons maintenant préciser les différents concepts présentés dans cet exemple.

2.1 Module `Tkinter`

Par défaut, Python ne permet les interactions que sur une console de terminal.

Pour permettre la construction d'une interface graphique, nous utilisons le module `Tkinter` qui fournit de nombreux constructeurs d'objets graphiques, et autres services.

Il y a deux façons de charger le module `Tkinter` :

- `import Tkinter` tous les noms de méthodes doivent être préfixé par `Tkinter`.
- `from Tkinter import *` importe tous les noms de méthodes du module `Tkinter`.

On voit ci-dessous, selon les cas, les différences dans l'écriture du code :

```
import Tkinter
fenetre = Tkinter.Tk()
etiquette = Tkinter.Label(fenetre)
champ = Tkinter.Entry(fenetre)
bouton = Tkinter.Button(fenetre)
...
```

```
from Tkinter import *
fenetre = Tk()
etiquette = Label(fenetre)
champ = Entry(fenetre)
bouton = Button(fenetre)
...
```

2.2 Objet, Classe, Constructeur, Instance, Propriété, Méthode

La classe d'un objet précise le concept qu'il représente. Un objet possède une structure interne qui mémorise des données sous forme de propriétés. Les messages auxquels est capable de répondre un objet permettent de définir son comportement.

Prenons l'exemple de l'étiquette dans notre programme page 3 :

```
1 etiquette = Label(fenetre)
2 etiquette["text"] = "Bonjour tout le monde"
3 etiquette.pack()
```

Ligne par ligne :

1. `Label` est un *constructeur* qui crée une *instance* c'est à dire un exemplaire d'objet de *classe* `Label`. La variable `etiquette` fait maintenant référence à cette instance.
2. on fixe la valeur de la *propriété* `text` de l'objet `etiquette` à `"Bonjour tout le monde"` à chaque *propriété* d'un objet peut ainsi être attribuée une *valeur*.
3. on envoie le *message* `pack()` à l'objet `etiquette`, ainsi, la *méthode* `pack` place le widget dans la fenêtre en fixant différentes propriétés géométriques de cet objet.

2.3 Callback

Les objets possèdent de très nombreuses propriétés qui caractérisent leur aspect (position couleur style etc.) et qui seront précisées section 3 page 6.

Certains widgets comme les boutons peuvent être contrôlés ou activés par l'utilisateur lors de l'exécution du programme. Il est donc nécessaire d'associer à ces objets un *callback* c'est à dire une fonction à exécuter sur un événement particulier.

L'instruction `bouton["command"] = faireBis` affecte la propriété `command` de l'objet `bouton` et lui donne pour valeur la fonction `faireBis` à appeler lors du clic sur le bouton.

2.4 Layout Manager

Dans notre programme page 3, nous avons simplement transmis le message `pack()` à chacun des widgets créés. On constate, dans ce cas, que la méthode `pack` a aligné ces widgets les uns en dessous des autres, centrés dans la fenêtre. On remarque même que les widgets restent centrés lorsqu'on change les dimensions de la fenêtre.

Ce placement par défaut a été effectué par le *Layout Manager* ou gestionnaire de position lors de l'appel de la méthode `pack`.

Le gestionnaire de position permet de disposer les widgets différemment et nous verrons plus en détail comment procéder section 5 page 11.

2.5 Boucle des événements

La dernière instruction de notre programme `fenetre.mainloop()` fait tout le reste.

La fenêtre est ouverte, les widgets y sont placés, et le programme tourne en boucle dans l'attente d'un événement extérieur.

Certains événements sont pris en compte par le système et nous n'avons pas à nous en préoccuper, comme :

- les déplacements de la fenêtre
- les changements des dimensions de la fenêtre
- la frappe des textes dans les champs de saisie

D'autres événements sont pris en compte par notre programme, comme l'action du bouton.

Le dernier événement pris en compte sera le clic sur la case de fermeture de la fenêtre qui a pour effet d'interrompre la boucle d'attente des événements et de faire disparaitre la fenêtre.

L'exécution du programme est alors terminée.

3 Objets `Tkinter`, Widgets

On va maintenant décrire ici les principaux widgets proposés par `Tkinter` et préciser leurs propriétés. Ce texte n'est pas un document de référence sur `Tkinter`. On se limitera ici aux widgets et aux propriétés les plus utiles pour construire rapidement une interface simple.

Mais, avant de décrire les widgets, on doit d'abord créer la fenêtre destinée à les contenir.

3.1 Fenêtre principale : `Tk`

Dans tous les exemples proposés, les widgets seront placés dans la fenêtre principale de l'application créée par les deux lignes de code suivantes :

```
from Tkinter import *
fenetre = Tk()
```

Il est possible de contrôler la présentation de la fenêtre sur l'écran de l'ordinateur. Par défaut, `"tk"` est écrit sur la barre de titre de la fenêtre qui s'ouvre avec une taille qui permet d'afficher tous les widgets créés.

Mais on peut redéfinir le titre de la fenêtre et fixer les dimensions minimales et maximales permises pour les changements de taille de la fenêtre, ainsi :

```
fenetre.title("Mon titre personnel")
fenetre.minsize(200, 100)
fenetre.maxsize(800, 400)
```

On peut même interdire la modification des dimensions de la fenêtre avec :

```
fenetre.resizable(width=False, height=False)
```

On rappelle que ces instructions construisent l'objet fenêtre de classe `Tk` mais que c'est seulement l'instruction `fenetre.mainloop()` qui ouvrira réellement la fenêtre et lancera la boucle d'attente des événements.

3.2 Étiquette : `Label`

Une étiquette est un widget de classe `Label` qui permet d'afficher un texte dans la fenêtre. Les principales propriétés d'une étiquette sont :

- `text` le texte affiché par l'étiquette
- `justify` la justification du texte, valeurs : `LEFT RIGHT CENTER`
- `background` ou `bg`, `foreground` ou `fg` les couleurs
  pour plus d'informations sur les couleurs, voir annexe C page 17
- `width` largeur de l'étiquette en nombre de caractères
- `height` hauteur de l'étiquette en nombre de lignes
- `borderwidth` ou `bd`, `relief` le style de l'encadrement de l'étiquette
  pour plus d'informations sur les encadrements, voir annexe B page 17

Voici, par exemple, comment placer dans le fenêtre le texte `"Bonjour"`sur fond jaune :

```
etiquette = Label(fenetre)
etiquette["text"] = "Bonjour"
etiquette["background"] = "yellow"
etiquette.pack()
```

3.3 Widget actif ou passif

Précisons dès maintenant la différence entre un widget *actif* et un widget *passif*, en utilisant l'exemple précédent.

On dira que l'étiquette `"Bonjour"` est active si ses propriétés peuvent changer pendant l'exécution du programme.

Par exemple, si on exécute les instructions suivantes en cours de programme :

```
etiquette["text"] = "Au revoir"
etiquette["text"] = "cyan"
```

Le texte et la couleur du fond changeront dynamiquement.

Ceci est possible, car à la création de l'étiquette, nous avons mémorisé l'objet dans la variable `etiquette`, ce qui nous permet d'y faire référence plus tard.

Par contre, si nous n'avons pas l'intention de modifier l'étiquette `"Bonjour"` dans la suite du programme, on dira que l'étiquette est passive. Il n'est donc pas nécessaire de conserver une variable pour la référencer.

Dans ce cas, la mise en place de l'étiquette dans l'interface peut se faire avec une instruction unique, sous cette forme :

```
(Label(fenetre, text="Bonjour", background="yellow")).pack()
```

On ajoute comme argument du constructeur `Label` les valeurs à affecter aux propriétés et on place immédiatement le widget dans la fenêtre en appelant la méthode `pack` sur l'objet créé, mis entre parenthèses. Dans ce cas, on ne conserve pas de référence vers le widget.

Cette méthode peut s'appliquer à tous les widget passifs d'une interface. Ceci peut simplifier l'écriture du code quand les widgets sont nombreux.

3.4 Bouton : `Button`

Un bouton est un widget de classe `Button` qui fournit à l'utilisateur un moyen de communiquer avec l'application avec un simple clic qui provoque une action programmée.
Les principales propriétés d'un bouton sont :

- `text` le texte affiché dans le bouton
- `justify` la justification du texte, valeurs : `LEFT RIGHT CENTER`
- `command` le *callback* ou fonction à appliquer lors du clic de l'utilisateur sur le bouton

Voici, par exemple, une instruction qui place un bouton dans la fenêtre :

```
(Button(fenetre, text="Bis", command=faireBis)).pack()
```

Attention : Dans ce cas, la fonction `faireBis` doit avoir été définie au préalable.

3.5 Champ de saisie de texte : `Entry`

Un champ de saisie de texte est un widget de classe `Entry` qui fournit à l'utilisateur une case dans laquelle il peut taper un texte avec le clavier.

Les propriétés les plus utiles d'un champ de saisie de texte sont :

- `width` qui permet de fixer la largeur du champ en nombre de caractères (20 par défaut)
- `textvariable` lien vers une variable de classe `StringVar`

Voici, par exemple, comment placer un champ de saisie de texte dans la fenêtre :

```
champ = Entry(fenetre, width=50)
champ.pack()
```

Il est impératif de mémoriser le widget de classe `Entry` dans une variable si on veux avoir accès, plus tard, au texte contenu dans le champ.
Ce texte est en effet récupérable à l'aide de la méthode `get` ainsi :

```
texte = champ.get()
```

Cependant pour faciliter l'accès au contenu du champ de saisie, il est conseillé de lui associer une variable chaîne, c'est à dire un objet de classe `StringVar` créé au préalable.

On obtient alors le code suivant :

```
texteSaisi = StringVar()
(Entry(fenetre, textvariable=texteSaisi)).pack()
```

On a maintenant accès facilement au contenu du champ de saisie aussi bien en lecture qu'en écriture à travers la variable `texteSaisi`.

On récupère la valeur du contenu du champ avec la méthode `get`, et on peut aussi en modifier le contenu avec la méthode `set`, ainsi :

```
texte = texteSaisi.get()
texteSaisi.set("nouveau texte")
```

Attention : Il ne faut pas confondre `texteSaisi` qui est une variable Python ordinaire qui référence l'objet de classe `StringVar` qui est une variable de contrôle au sens de `Tkinter` et qui permet d'accéder dynamiquement à une propriété du widget auquel elle est liée.

On remarque dans ce cas qu'il est inutile de conserver une variable Python qui référence le champ de saisie, la variable de contrôle de classe `StringVar` suffit.

3.6 Case à cocher : `Checkbutton`

Une case à cocher est un widget de classe `Checkbutton` qui fournit à l'utilisateur un moyen de définir une valeur booléenne (vrai ou faux) avec un simple clic.
Les principales propriétés d'une case à cocher sont :

- `text` le texte affiché à côté de la case
- `justify` la justification du texte, valeurs : `LEFT RIGHT CENTER`
- `variable` lien vers une variable de classe `BooleanVar`

Voici, par exemple, comment placer une case à cocher dans la fenêtre :

```
etatSaisi = BooleanVar()
(Checkbutton(fenetre, text="Accepter", variable=etatSaisi)).pack()
```

On récupère l'état de la case à cocher avec la méthode `get` (`True` si la case est cochée et `False` sinon), et on peut aussi en fixer l'état par programme avec la méthode `set`, ainsi :

```
etat = etatSaisi.get()
etatSaisi.set(True)    #  ce qui coche la case
```

3.7 Cercle d'option : `Radiobutton`

Les cercles d'options sont des widgets de classe `Radiobutton` qui sont associés en groupes pour permettre à l'utilisateur de choisir parmi une liste de valeurs deux à deux exclusives.

Les principales propriétés d'un cercle d'option sont :

- `text` le texte affiché à côté du cercle
- `value` la valeur associée à cette option
- `variable` lien vers une variable de classe `IntVar` ou `StringVar`

Les différents cercles d'options d'un même groupe doivent être liés à la même variable.

Voici, par exemple, comment placer trois cercles d'options groupé dans la fenêtre :

```
choixSaisi = IntVar()
(Radiobutton(fenetre, text="Option 1", value=1, variable=choixSaisi)).pack()
(Radiobutton(fenetre, text="Option 2", value=2, variable=choixSaisi)).pack()
(Radiobutton(fenetre, text="Option 3", value=3, variable=choixSaisi)).pack()
```

On récupère la valeur de l'option avec la méthode `get`, mais on peut aussi fixer l'option par programme avec la méthode `set`, ainsi :

```
choix = choixSaisi.get()
choixSaisi.set(2)
```

Cette dernière instruction sélectionne le cercle `"Option 2"` et désélectionne les autres.

3.8 Zone de dessin : `Canvas`

Une zone de dessin est un widget de classe `Canvas` qui permet de placer dans la fenêtre une zone rectangulaire dans laquelle des méthodes spécifiques permettent de tracer par programme, les lignes des rectangles des ovales ou des polygones.

Les principales propriétés d'une zone de dessin sont :

- `background` ou `bg` la couleur du fond
  pour plus d'informations sur les couleurs, voir annexe C page 17
- `width` et `height` largeur et hauteur de la zone de dessin
  pour plus d'informations sur les dimensions, voir annexe A page 17

Les méthodes de dessin sur un `Canvas` sont décrites en détail annexe D page 18

3.9 Cadre : `Frame`, `LabelFrame`

Un cadre est un widget de classe `Frame` ou `LabelFrame` qui permet de placer une zone rectangulaire vide dans la fenêtre. Cette zone peut contenir d'autres widgets, ce qui permet de structurer la disposition des widgets dans la fenêtre.
Les principales propriétés d'un cadre sont :

- `text` le texte affiché comme titre du cadre, uniquement pour la classe `LabelFrame`
- `background` ou `bg`, `foreground` ou `fg` les couleurs
  pour plus d'informations sur les couleurs, voir annexe C page 17
- `width` et `height` largeur et hauteur du cadre
  pour plus d'informations sur les dimensions, voir annexe A page 17
- `borderwidth` ou `bd`, `relief` le style du cadre
  pour plus d'informations sur les encadrements, voir annexe B page 17

Nous verrons dans le paragraphe suivant comment utiliser les cadres pour disposer de manière satisfaisante les widgets dans la fenêtre.

4 Fenêtres de dialogues auxiliaires

Pour interagir avec l'utilisateur, il est parfois utile d'ouvrir une fenêtre auxiliaire de dialogue qui affiche une information à laquelle peut répondre l'utilisateur.

Pour utiliser des fenêtres de dialogue auxiliaires il est nécessaire de charger un module complémentaire avant utilisation :

```
import tkMessageBox
```

Les méthodes proposées par le module `tkMessageBox` ouvrent une fenêtre spécifique de dialogue auxiliaire que l'utilisateur peut refermer après consultation par un clic sur l'un des boutons disponibles. Pour chacune de ces méthodes, le premier argument est le titre de la fenêtre, le second est le message affiché :

- `showinfo` affichage d'une information, fermeture après lecture par clic sur `"OK"`
- `askokcancel` demande de confirmation et retourne comme valeur :
  `True` après un clic sur `"OK"` `False` après un clic sur `"Cancel"`
- `askquestion` pose une question et retourne comme valeur :
  `"yes"` après un clic sur `"Yes"` `"no"` après un clic sur `"No"`

Exemples d'instructions utilisant des dialogues auxiliaires :

```
tkMessageBox.showinfo("Météo", "Il fait beau aujourd'hui")

if tkMessageBox.askokcancel("ATTENTION", "On efface vraiment tout ?"):
    toutEffacer()

if tkMessageBox.askquestion("Impôts", "Etes-vous riche ?") == "no":
    faireUneDeclarationSimple()
else:
    faireUneDeclarationISF()
```

5 Construction d'une interface complexe

On va présenter maintenant quelques propriétés avancées du Layout Manager qui permettent de construire des interfaces plus complexes.

Nous décrirons chaque méthode de construction avec un exemple simple commenté qui utilise les diverses possibilités du module `Tkinter`.

Les deux principale méthodes de placement des widgets dans la fenêtre sont :

- la méthode `pack` qui permet de placer les widgets les uns à la suite des autres en respectant certaines règles que l'on décrira au paragraphe 5.1 page 11.
- la méthode `grid` qui permet de placer les widgets dans une grilles composée de lignes et de colonnes selon certaines règles que l'on décrira au paragraphe 5.2 page 14.

Il faut choisir une des deux méthodes et éviter d'utiliser `pack` et `grid` simultanément.

5.1 Avec la méthode `pack` : `"Pack-Tkinter.py"`

La méthode `pack` qui permet de placer les widgets les uns à la suite des autres et dispose de plusieurs options qui permettent de préciser le mode d'alignement des widgets.

Cette méthode peut être utilisée en même temps que les cadres de classe `Frame` ou `LabelFrame` pour placer les éléments de l'interface de façon structurée.

Les options de la méthode `pack` sont :

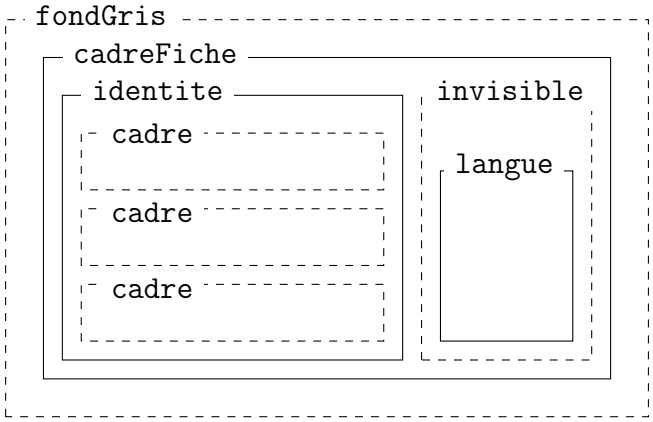
- `side` indique dans quel sens s'alignent les widgets :
  - `TOP` le défaut, du haut vers le bas, centré horizontalement dans le parent
  - `LEFT` de la gauche vers la droite, centré verticalement dans le parent

`RIGHT` de la droite vers la gauche, centré verticalement dans le parent
`BOTTOM` du bas vers le haut, centré horizontalement dans le parent

- `fill` indique comment le widget remplit l'espace disponible dans le parent :
  `NONE` par défaut, aucun remplissage
  `X` remplissage horizontal
  `Y` remplissage vertical
  `BOTH` remplissage dans les deux direction
- `expand=0` expansion interdite, par défaut
- `expand=1` expansion qui force à occuper toute la place disponible dans le parent
- `padx` nombre de pixels qui fixe l'espace vide horizontalement autour du widget
- `pady` nombre de pixels qui fixe l'espace vide verticalement autour du widget

Nous allons maintenant présenter un exemple pas à pas et commenter les options utilisées. Nous voulons faire une fiche de saisie d'identité comme celle-ci :

Nous allons emboîter plusieurs cadres selon le schémas présenté ci-dessous :



Les noms affichés sur la figure sont ceux des variables du programme qui référencent les cadres utilisés. Les cadres en pointillés ne sont pas visible sur l'interface, et les widgets placés dans ces cadres n'ont pas été représentés. (*ce code est dans le fichier* `Pack-Tkinter.py`)

- Créons d'abord une fenêtre pour y placer notre fiche :

```
1 from Tkinter import *
2 fenetrePrincipale = Tk()
3 fondGris = Frame(fenetrePrincipale, background="gray")
4 fondGris.pack(fill=BOTH, expand=1)
5 cadreFiche = Frame(fondGris, borderwidth=4, relief=SUNKEN)
6 cadreFiche.pack(padx=15, pady=15)
```

Ligne 2 on crée la fenêtre principale.

Ligne 3 on crée un cadre `fondGris` ayant pour parent `fenetrePrincipale`.

Ligne 4 on le place dans la fenêtre principale en le forçant à occuper toute la place disponible.

même si l'utilisateur change les dimensions de la fenêtre

Ligne 5 on crée un cadre `cadreFiche` ayant pour parent le cadre `fondGris`. La couleur du fond sera blanche, par défaut, mais il sera encadré d'une bordure de 4 pixels avec relief en creux.

Ligne 6 on le place par défaut, centré en haut, mais à 15 pixels des bords horizontalement et verticalement. *Il ne changera pas de taille même si l'utilisateur change les dimensions de la fenêtre, mais restera centré en haut de la fenêtre.*

• Partageons maintenant notre fiche en deux horizontalement :

```
7  identite = LabelFrame(cadreFiche, text="Identité")
8  identite.pack(side=LEFT, padx=5, pady=5)
9  invisible = Frame(cadreFiche)
10 invisible.pack(side=LEFT, fill=Y, padx=5, pady=5)
```

Ligne 7 on crée un cadre `identite` de classe `LabelFrame` avec étiquette `"Identité"` ayant pour parent `cadreFiche`.

Ligne 8 on le place à gauche bordé d'un espace vide de 5 pixels.

Ligne 9 on crée un cadre `invisible` de classe `Frame` ayant aussi pour parent `cadreFiche`.

Ligne 10 on le place aussi à gauche à 5 pixels des bords, mais cette fois `fill=Y` précise qu'il doit occuper toute la place verticalement, même si son contenu est moins haut que celui du cadre `identite` précèdent.

• Construisons maintenant notre fiche d'identité. Pour cela on va créer successivement 3 lignes à l'aide de la variable auxiliaire cadre qui servira de référence à 3 cadres successif, de classe `Frame`, invisibles ayant pour parent `identite` :

```
11 cadre = Frame(identite)
12 cadre.pack(fill=X, padx=2, pady=2)
13 civilite = StringVar()
14 (Radiobutton(cadre, text="M", variable=civilite, value="M")).pack(side=LEFT)
15 (Radiobutton(cadre, text="Me", variable=civilite, value="Me")).pack(side=LEFT)
16 (Radiobutton(cadre, text="Melle", variable=civilite, value="Melle")).pack(side=LEFT)
17 cadre = Frame(identite)
18 cadre.pack(fill=X, padx=2, pady=2)
19 nom = StringVar()
20 (Entry(cadre, textvariable=nom)).pack(side=RIGHT)
21 (Label(cadre, text="Nom")).pack(side=RIGHT)
22 cadre = Frame(identite)
23 cadre.pack(fill=X, padx=2, pady=2)
24 prenom = StringVar()
25 (Entry(cadre, textvariable=prenom)).pack(side=RIGHT)
26 (Label(cadre, text="Prénom")).pack(side=RIGHT)
```

Lignes 11 et 12 on crée un cadre invisible pour la *première ligne* ayant pour parent `identite` et on le place en haut par défaut à 2 pixels des bords avec remplissage horizontal du parent, ce qui permettra d'aligner ses fils à gauche.

Ligne 13 on crée la variable de contrôle `civilite` de classe `StringVar` pour les options.

Lignes 14 à 16 on crée 3 cercles d'options en les liant à la même variable de contrôle `civilite`, en fixant leur attribut `value` et en les plaçant successivement à gauche.

Lignes 17 et 18 on crée un cadre invisible pour la *seconde ligne* ayant pour parent `identite` et on le place par défaut à 2 pixels des bords avec remplissage horizontal du parent, ce qui permettra d'aligner ses fils à droite.

Ligne 19 on crée la variable de contrôle `nom` de classe `StringVar` pour la saisie du nom.

Lignes 20 et 21 on crée le champ de saisi lié à la variable de contrôle `nom`, puis une étiquette et on les placent successivement cadrés à droite.

Lignes 24 à 26 on procède de la même façon pour le prénom.

• Construisons pour terminer le cadre de choix des langues notre fiche :

```
27 langue = LabelFrame(invisible, text="Langue")
28 langue.pack(side=BOTTOM)
29 francais = IntVar()
30 (Checkbutton(langue, text="Français", variable=francais, justify=LEFT)).pack(fill=BOTH)
31 anglais = IntVar()
32 (Checkbutton(langue, text="Anglais", variable=anglais, justify=LEFT)).pack(fill=BOTH)
33 autre = IntVar()
34 (Checkbutton(langue, text="autre", variable=autre, justify=LEFT)).pack(fill=BOTH)
```

Ligne 27 on crée un cadre `identite` de classe `LabelFrame` avec étiquette `"Langue"` ayant pour parent notre cadre `invisible`.

Ligne 28 on la place en bas de son parent, qui, rappelons-le, occupe la place verticalement, même si son contenu est moins haut que celui du cadre `identite` précédent.

Lignes 29 à 34, pour chaque case d'option, on définit une variable de contrôle de classe `StringVar`, on crée un widget de classe `Checkbutton` et on le place en forçant le remplissage. Pour chaque case d'option on fixe `justify=LEFT` pour que le texte s'aligne correctement.

• On termine en lançant la boucle d'attente des événements :

```
35 fenetrePrincipale.mainloop()
```

5.2 Avec la méthode `grid` : `"Grid-Tkinter.py"`

La méthode `grid` permet de réaliser rapidement des interfaces simples en plaçant les widgets dans une grilles composée de lignes et de colonnes.

Cette méthode dispose de plusieurs options qui permettent de préciser les coordonnées de la cellule dans laquelle est placé le widget et son mode de placement.

Les dimensions des cellules de la grilles sont fixée en largeur par l'élément le plus large pour chaque colonne, et en hauteur par l'élément le plus haut pour chaque ligne.

En général on place un widget unique dans chaque cellule de la grille, mais certaines options permettent d'étendre un widget sur plusieurs cellules contiguës.

Les options de la méthode `grid` sont :

- `row` fixe le numéro de la ligne de la cellule, 0 pour la première ligne.
- `column` fixe le numéro de la colonne de la cellule, 0 pour la première colonne.
- `rowspan` fixe le nombre de cellules contiguës occupée par le widget sur une colonne.

  exemple : `widget.grid(row=0, column=2, rowspan=3)`

  place le widget sur les lignes 0, 1, et 2 de la colonne 2.

– `columnspan` fixe le nombre de cellules contiguës occupée par le widget sur une ligne.
exemple : `widget.grid(row=0, column=2, columnspan=3)`
place le widget sur les colonnes 2, 3, et 4 de la ligne 0.
– `padx` le nombre de pixels qui fixe l'espace horizontal autour de la cellule.
– `pady` le nombre de pixels qui fixe l'espace vertical autour de la cellule.
– `sticky` quand un widget est plus petit que la cellule où il est, on peut l'étendre en le faisant coller aux bords de la cellule en fixant ainsi les points d'ancrage :
`NW N NE` nord-ouest, nord, nord-est
`W E` ouest, est
`SW S SE` sud-ouest, sud, sud-est
exemple : `widget.grid(row=1, column=2, sticky=W+E)`
étend le widget horizontalement (collé à l'ouest et à l'est).
exemple : `widget.grid(row=1, column=2, sticky=N+S)`
étend le widget verticalement (collé au nord et au sud).

Nous allons maintenant donner un exemple qui montre avec de simples étiquettes comment placer des éléments avec la méthode `grid`. Pour cela nous allons placer dix widgets étiquetés de `"Zéro"` à `"9"` sur une grille de quatre lignes et quatre colonnes. Certaines étiquettes seront placées dans une cellules unique, et certaines seront étendues sur plusieurs cellules contiguës.

Pour visualiser la géométrie de la grille, on a fixé par `background="gray"` la couleur du fond de l'étiquette. De plus pour bien séparer les cellules de la grille, on les a espacées de 2 pixels par `padx=2, pady=2` (*ce code est dans le fichier* `Grid-Tkinter.py`)

Nous donnons simplement le code suivant qu'il faudra regarder attentivement :

```
1  from Tkinter import *
2  fenetrePrincipale = Tk()

3  widget0 = Label(fenetrePrincipale, bg='gray', text="Zéro")
4  widget0.grid(row=0, column=0, padx=2, pady=2)
5  widget1 = Label(fenetrePrincipale, bg='gray', text="Un")
6  widget1.grid(row=0, column=1, padx=2, pady=2)
7  widget2 = Label(fenetrePrincipale, bg='gray', text="Deux")
8  widget2.grid(row=0, column=2, columnspan=2, sticky=W+E, padx=2, pady=2)
9  widget3 = Label(fenetrePrincipale, bg='gray', text="Trois")
10 widget3.grid(row=1, column=0, padx=2, pady=2)
11 widget4 = Label(fenetrePrincipale, bg='gray', text="Quatre")
12 widget4.grid(row=1, column=1, rowspan=3, sticky=N+S, padx=2, pady=2)
13 widget5 = Label(fenetrePrincipale, bg='gray', text="5")
14 widget5.grid(row=1, column=2, padx=2, pady=2)
15 widget6 = Label(fenetrePrincipale, bg='gray', text="6")
16 widget6.grid(row=1, column=3, padx=2, pady=2)
17 widget7 = Label(fenetrePrincipale, bg='gray', text="7")
18 widget7.grid(row=2, column=0, padx=2, pady=2)
19 widget8 = Label(fenetrePrincipale, bg='gray', text="8")
20 widget8.grid(row=2, column=2, columnspan=2, rowspan=2, sticky=NW+SE, padx=2, pady=2)
21 widget9 = Label(fenetrePrincipale, bg='gray', text="9")
22 widget9.grid(row=3, column=0, sticky=NW+SE, padx=2, pady=2)

23 fenetrePrincipale.mainloop()
```

Nous remarquons que, par défaut, chaque widget est centré sur sa cellule et n'occupe que la place correspondant à son texte.

Par contre, si l'option `sticky` est utilisée, le widget est étendu sur toute la largeur ou la hauteur de la cellule selon le cas :

- horizontalement pour `widget2` ligne 8 (collé à l'ouest et à l'est)
- verticalement pour `widget4` ligne 12 (collé au et au sud)
- dans les deux sens pour `widget8` et `widget9` lignes 20 et 22 (`sticky=NW+SE`)

Remarque : Si l'utilisateur change les dimension de la fenêtre, les widgets restent dans le coin en haut à gauche d la fenêtre.

Cependant, la grille peut occuper la totatlité de la fenêtre à condition d'accorder un *poids* (`weight`) à chaque ligne et à chaque colonne de la grille.

Pour cela, on utilisera les méthodes `rowconfigure` et `columnconfigure`.

Par exemple, si `fenetrePrincipale` contient deux lignes :

```
fenetrePrincipale.rowconfigure(0, weight=1)
fenetrePrincipale.rowconfigure(1, weight=3)
```

Même après changement des dimensions de la fenêtre la première ligne occupera 1/4 et la seconde ligne occupera 3/4 de l'espace disponible.

Dans notre exemple précédent, si nous voulons que toutes les cellules de la grille soient identiques, nous fixerons pour chaque ligne et chaque colonne le même poids.

Pour cela, avant la ligne 23, la dernière du programme, insérez le code suivant :

```
for li in range(4):
    for co in range(3):
        fenetrePrincipale.rowconfigure(li, weight=1)
        fenetrePrincipale.columnconfigure(co, weight=1)
```

Modifiez les dimensions de la fenêtre principale pour vérifier le résultat.

6 Informations complémentaires

On pourra consulter le site ISN, « Spécialité Informatique et Sciences du Numérique »

http://mathmac.free.fr/

À la rubrique « Interface »

http://mathmac.free.fr/-interface.html

Au sujet de `Tkinter`, on trouvera des informations très complètes (en anglais) sur le site d'une université américaine « New Mexico Tech Computer Center »

http://infohost.nmt.edu/tcc/help/pubs/tkinter/

A Dimension

Certaines propriétés de widget attendent pour valeur une dimension :
- `width` : la largeur du widget
- `height` : la hauteur du widget
- `borderwidth` ou `bd` en abrégé : l'épaisseur de l'encadrement du widget

La dimension peut être définie comme :
- soit un nombre entier qui représente un nombre de pixels
- soit une chaîne de caractères : `"2c"` *pour* 2 *centimètres* ou `"8m"` *pour* 8 *millimètres*

La méthode `geometry` à la fenêtre principale permet de définir simultanément les dimensions de la fenêtre ou sa position sur l'écran sous la forme d'une chaîne de caractères :

`"`<largeur>`x`<hauteur>`"` et/ou `"+`<horizontal>`+`<vertical>`"`

Par exemple, après l'instruction suivante :

```
| fenetrePrincipale.geometry("500x300+250+50")
```

La fenêtre principale fera 500 pixels de large sur 300 pixels de haut et lors de son ouverture elle sera placée sur l'écran à 250 pixels du bord de gauche et 50 pixels du haut de l'écran.

```
| fenetrePrincipale.geometry("+250+50")
```

Ici elle sera simplement placée comme précédemment, mais avec ses dimensions par défaut.

B Encadrement

Par défaut, les widgets sont des rectangles plats. On peut leur donner du relief en précisant la largeur de leur bordure. Pour cela on utilise les propriétés :
- `borderwidth` ou `bd` en abrégé : l'épaisseur de l'encadrement du widget
- `relief` le type de l'encadrement du widget

Les valeurs de la propriété `relief` sont les constantes suivantes :
- `FLAT` sans bordure (défaut)
- `RAISED` plaque en relief
- `SUNKEN` plaque en creux
- `GROOVE` contour en creux
- `RIDGE` contour en relief

C Couleur

Certaines propriétés de widget attendent pour valeur une couleur :
- `background` ou `bg` en abrégé : la couleur du fond du widget
- `foreground` ou `fg` en abrégé : la couleur du texte ou de l'avant plan du widget

La couleur est définie comme une chaîne de caractères :
- soit un nom prédéfini : `"black"` `"white"` `"red"` `"green"` `"blue"` `"cyan"` `"yellow"`
- soit une définition de la forme `"#rgb"` composée de 3 chiffres héxadécimaux
  par exemple : noir `"#000"`, blanc `"#FFF"`, gris clair `"#CCC"` ou mauve `"#C6F"`

D Dessiner dans un `Canvas` : `"Canvas-Tkinter.py"`

Nous décrivons ici uniquement quelques méthodes de base utiles pour engendrer un dessin par programme dans un widget de classe `Canvas`.

Les principales méthodes sont :
- `create_line` pour dessiner une ligne constituées de segments de droites
- `create_polygon` pour dessiner un polygone fermé constitué de segments de droites
- `create_rectangle` pour dessiner un rectangle
- `create_oval` pour dessiner une ellipse
- `create_arc` pour dessiner un arc d'ellipse

Nous allons préciser les paramètres de chacune de ces méthodes transmises comme message à `dessin` un widget de classe `Canvas` de fond gris clair installé dans la fenêtre ainsi :

```
dessin = Canvas(fenetre, width=400, height=200, bg="#EEE")
dessin.pack()
```

Le code de cet exemple est dans le fichier `Canvas-Tkinter.py`

D.1 Ligne avec `create_line`

La méthode `create_line` attend comme paramètres une suite de couples de cordonnées de points plus des options de tracé comme :
- `fill` fixe la couleur de la ligne (black par défaut)
- `width` fixe l'épaisseur de la ligne (1 pixel par défaut)
- `dash` définit un mode de pointillés *par exemple* : `dash=(2,3,2,5)` trace successivement un trait de 2 pixels, un espace de 5, un trait de 2, un espace de 5

Exemple :

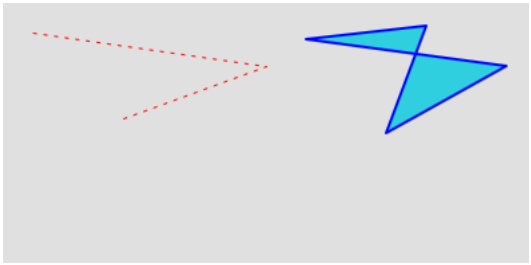
```
dessin.create_line(25, 25, 200, 50, 90, 90, fill="red", dash=(3,5))
```

D.2 Polygone avec `create_polygon`

La méthode `create_polygon` attend comme paramètres une suite de couples de cordonnées de points plus des options de tracé comme :
- `outline` fixe la couleur de la ligne (black par défaut)
- `fill` fixe la couleur de remplissage du polygone (white par défaut)
- `width` fixe l'épaisseur de la ligne (1 pixel par défaut)
- `dash` définit un mode de pointillés *par exemple* : `dash=(2,3,2,5)` trace successivement un trait de 2 pixels, un espace de 5, un trait de 2, un espace de 5

Exemple :

```
dessin.create_polygon(230, 30, 380, 50, 290, 100, 320, 20,
                      fill="#3DE", width=2, outline="blue")
```



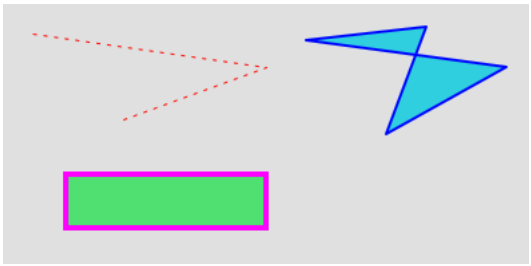
D.3 Rectangle avec `create_rectangle`

La méthode `create_rectangle` attend comme paramètres deux couples de cordonnées de points, le coin en haut à gauche et le coin en bas à droite, plus des options de tracé comme :

- `outline` fixe la couleur de la ligne (black par défaut)
- `fill` fixe la couleur de remplissage du rectangle (white par défaut)
- `width` fixe l'épaisseur de la ligne (1 pixel par défaut)
- `dash` définit un mode de pointillés *par exemple* : `dash=(2,3,2,5)` trace successivement un trait de 2 pixels, un espace de 5, un trait de 2, un espace de 5

Exemple :

```
dessin.create_rectangle(50, 130, 200, 170,
                        outline="magenta", fill="#5E7", width=4)
```



D.4 Ellipse avec `create_oval`

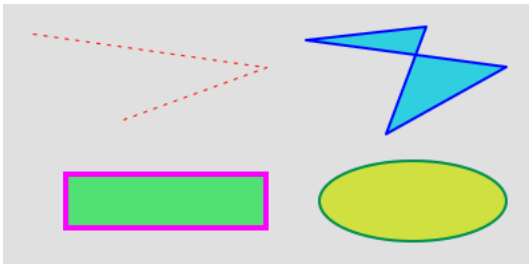
La méthode `create_oval` attend comme paramètres deux couples de cordonnées de points, le coin en haut à gauche et le coin en bas à droite du rectangle contenant l'ellipse, plus des options de tracé comme :

- `outline` fixe la couleur de la ligne (black par défaut)
- `fill` fixe la couleur de remplissage de l'ellipse (white par défaut)
- `width` fixe l'épaisseur de la ligne (1 pixel par défaut)
- `dash` définit un mode de pointillés *par exemple* : `dash=(2,3,2,5)` trace successivement un trait de 2 pixels, un espace de 5, un trait de 2, un espace de 5

Remarque : Pour obtenir un cercle les coordonnées doivent définir les coins en haut à gauche et en bas à droite d'un carré.

Exemple :

```
dessin.create_oval(240, 120, 380, 180, outline="#095", fill="#DE4", width=2)
```



D.5 Arc avec `create_arc`

La méthode `create_arc` attend comme paramètres deux couples de cordonnées de points, coin en haut à gauche et le coin en bas à droite du rectangle contenant l'ellipse qui sert de support de l'arc, plus des options de tracé comme :

- `outline` fixe la couleur de la ligne (black par défaut)
- `fill` fixe la couleur de remplissage de l'arc d'ellipse (white par défaut)
- `width` fixe l'épaisseur de la ligne (1 pixel par défaut)
- `dash` définit un mode de pointillés *par exemple* : `dash=(2,3,2,5)` trace successivement un trait de 2 pixels, un espace de 5, un trait de 2, un espace de 5
- `start` fixe l'angle de départ de l'arc (0 ° par défaut départ à droite à l'horizontale)
- `extent` fixe l'angle de l'arc (45 ° par défaut dans le sens trigonométrique positif)
- `style` fixe le style de l'arc selon les constantes suivante :
  - `PIESLICE` valeur par défaut : les extrémités de l'arc sont reliées au centre
  - `CHORD` les extrémités de l'arc sont reliées entre elles
  - `ARC` les extrémités de l'arc ne sont pas reliées (pas de remplissage possible)

Remarque : Pour obtenir un arc de cercle les coordonnées doivent définir les coins en haut à gauche et en bas à droite d'un carré.

Exemple :

```
dessin.create_arc(180, 60, 280, 160, fill="yellow", start=45, extent=120)
```